PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
G06K 9/00, 9/46, 9/48
G06K 9/34, 9/36

A1

(11) International Publication Number: **WO 92/11609**

(43) International Publication Date: 9 July 1992 (09.07.92)

(21) International Application Number: PCT/US91/09477

(22) International Filing Date: 17 December 1991 (17.12.91)

(30) Priority data:
632,232 19 December 1990 (19.12.90) US

(71) Applicant: OPTICAL SPECIALITIES, INC. [US/US]; 4281 Technology Drive, Fremont, CA 94538 (US).

(72) Inventor: MAPLES, James ; 762 Allen Court, Palo Alto, CA 94303 (US).

(74) Agent: BARR, Robert, A.; Townsend and Townsend, One Market Plaza - 2000 Steuart Tower, San Francisco, CA 94105 (US).

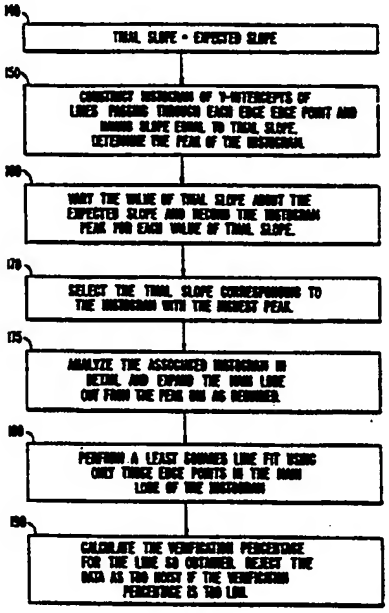
(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, LU (European patent), MC (European patent), NL (European patent), SE (European patent).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR AUTOMATED MEASUREMENT OF LOCATION AND SEPARATION OF LINEAR EDGES

(57) Abstract

A method and apparatus for detecting colinear edge points and measuring the distance between two edges in a digitized image. An image is captured (16) and digitized (20). The pixels corresponding to edge points are identified and processed to select colinear edge points by finding the straight line passing through the maximum number of edge points. The search for this line is performed by starting with an expected slope value (14a) and varying the slope by an amount corresponding to a predetermined angular increment. For each test slope, the y-intercepts are calculated (15a) for the straight lines passing through each edge point, and the best slope/intercept pair is determined. The points lying on or near the line defined by the best slope/intercept pair are selected and used to generate a straight line. The process is repeated for two edges and the distance between the two lines is measured.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MN	Mongolia
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Faso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GN	Guinea	NL	Netherlands
BJ	Benin	GR	Greece	NO	Norway
BR	Brazil	HU	Hungary	PL	Poland
CA	Canada	IT	Italy	RO	Romania
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU+	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE*	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		

\+ Any designation of "SU" has effect in the Russian Federation. It is not yet known whether any such designation has effect in other States of the former Soviet Union.

METHOD AND APPARATUS FOR AUTOMATED MEASUREMENT OF LOCATION AND SEPARATION OF LINEAR EDGES

BACKGROUND OF THE INVENTION

This invention relates generally to the field of automated measurement techniques and, more specifically, it relates to a method and apparatus for detecting collinear edge points in a digitized image and measuring the distance between two virtually parallel edges.

In many applications, it is necessary to measure the distance between two straight lines. For example, the measurement of the width of a strip of material on a semiconductor device may require a measurement of the distance between the substantially parallel edges. For another example, registration of two layers of a semiconductor product may be accomplished by aligning two images of known rectangular dimension, which requires a measurement of the distance between several pairs of parallel horizontal and vertical lines.

Automated systems exist for performing these measurements by capturing an image of the device,

digitizing the image, and analyzing the digitized image. The analysis of the image requires identifying the points in the image which represent the edges, "fitting" a straight line to each edge, and then determining the separation between the two lines.

The process of fitting a line to the edge points is complicated by the fact that, regardless of the method used to identify the edge points, there will be some points that do not correspond to the "true" line. These points are caused by noise in the image, dirt on the article being measured, or imperfections (bumps) on the edge. Known line-fitting methods, such as the ordinary least squares method, will produce a line which is a "best" fit to all of the points, but which, because of the inclusion of non-collinear points, can be an inadequate representation of the "true" line.

There are other methods, based on the Hough Transform disclosed in U.S. Patent No. 3,069,654, which detect collinear points by mapping the image points into lines in slope/intercept space. The slope/intercept space is quantized into cells, and the cell with the maximum number of lines passing through it corresponds to the slope and intercept of the straight line through the collinear points. The utility of this method is limited by the unbounded nature of the possible slope and intercept values, and the method does not work well at all for the common case of vertical lines, where the slope is infinite. An improvement on this method uses an angle-radius parameterization, thereby avoiding the problem of the unboundedness of the slope. Each point (x,y) in the image is mapped into a sinusoidal curve in angle-radius space defined by the equation:

$$\rho = x \cos \theta + y \sin \theta, \text{ where } 0 \leq \theta \leq \pi.$$

Such mapping requires large amounts of computational resources because it requires repeated calculation of transcendental trigonometric functions.

SUMMARY OF THE INVENTION

The invention provides a method and apparatus for detecting collinear edge points and measuring the distance between two edges in a digitized image without repeated calculation of transcendental functions. An image is captured and digitized. The edge points in the digitized image are identified and processed to select collinear edge points by finding the straight line passing through the maximum number of edge points. The search for this line is performed by starting with an expected slope value and varying the slope by an amount corresponding to a predetermined angular increment. For each test slope, the y-intercepts are calculated for the straight lines passing through each edge point, and the best slope/intercept pair is determined. The points lying on or near the line defined by the best slope/ intercept pair are selected and used to generate a straight line. The process is repeated for two edges and the distance between the two lines is measured.

A thorough understanding of this invention will be provided by the following description of the preferred embodiment and the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of an apparatus for practicing the invention;

Figs. 2A-2B are a flow chart of the method used in the preferred embodiment to process edge data;

Fig. 3 is a diagram of an edge and the resulting grey scale representation thereof; and

Fig. 4A-4C are diagrams of trial slopes applied to edge points and the histograms resulting from these trials.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to Fig. 1, a system 10 for performing measurements according to the invention is shown. A microscope 12 is focused on an area of the subject, which in this application is a semiconductor wafer 14. A camera 16 captures an image from microscope 12 and passes the image in electronic form to an image digitizer 20. Digitizer 20 converts the image to an ordered array of digital picture elements (pixels) and transmits the digital data to computer 22 for processing. In this embodiment, each pixel is an 8-bit "grey-scale" representation of a portion of the area being viewed, with a value in the range 0-255. A keyboard 24 and a display screen 26 are coupled to computer 22 for input and output of parameters and commands. Computer 22, which in this embodiment is a general purpose computer using an 80386 processor and an 80387 math co-processor (both available from Intel Corp. of Santa Clara, California), is coupled to control circuits 26. Control circuits 26 contain circuitry for controlling the parameters of the image capture system, such as focus, lighting (illumination), magnification, wafer position, etc.

The image obtained from microscope 12 and camera 16 is filtered and analyzed to determine the location, size and spacing of various features on semiconductor wafer 14. Many of the features that are analyzed require measuring the distance between two straight edges. This is accomplished by determining where in the image the two edges are, fitting a line to each edge, and then determining the separation between

the two lines. The line fitting process results in a mathematical characterization of a line which approximates the true edge.

Computer 22 analyzes the digitized image data to determine which pixels in the image correspond to points on an edge. Several edge-detection techniques are available for this purpose. In this embodiment, the pixels in each scan line are examined to determine the amount of contrast between adjacent pixels, i.e., the change in grey scale values. When the grey scale value crosses a predetermined threshold value with a predetermined amount of "rise" (gradient), this indicates a potential edge point. The threshold and rise amount can be calibrated for a particular application in accordance with the other viewing system parameters. The edge points are calculated to sub-pixel accuracy by interpolating between actual pixel coordinates. The resulting floating-point edge coordinates are stored in a list of edge points for further analysis.

Regardless of the methods used for filtering the image and locating edge points, the points that are located will not be collinear; there is always some noise in the image. In extreme cases, the image can be corrupted by dirt on the wafer, by slight imperfections in the formation of the semiconductor features, or other problems. These non-collinear points can distort any attempt to find a line that corresponds to the "true" edge. The invention provides a method for compensating for imperfections and obtaining reliable, reproducible line fitting results.

Referring now to Figs. 2A-2B, a flow chart 100 illustrates the method for processing the edge point data to find the line which best represents the true edge. The method illustrated in Figs. 2A-2B is implemented by the computer program supplied herewith as Appendix A,

which is executed on the computer system shown and described hereinabove with reference to Fig. 1. The computer program is written in the "C" programming language. In the following discussion, reference will be made to specific portions of the program, to enable a precise understanding of the preferred embodiment of the present invention. It will be understood, however, that those skilled in the art could implement this invention in many forms, including other programming languages or in hardware logic circuitry.

The input to the computer program of Appendix A is the floating point edge data, representing each edge point by its x-y coordinates. The output will be the slope and y-intercept defining a line according to the equation $y = mx + b$, along with a verification percentage for the line. The verification percentage is a measure of the proportion of the points which are on or close to the line.

The data is analyzed in stages, by making a quick initial estimate of the line and determining whether further processing is required. If the initial estimate is not sufficiently accurate, then the edge data is further analyzed to determine which points correspond to true edge points and which points correspond to noise or imperfections. Then a line is fit to the true edge points.

Referring to Fig. 2A, in Step 110, the edge points are determined by the method described above. At Step 120, a standard "least squares" algorithm is applied to the edge point data to find the line which is the "best" fit to all of the supposed edge points. All of the edge points are included in this calculation. In the common case, where all of the edge points are "good" (they are all essentially on a straight line) this method will yield a good fit. If however the line is corrupted

by a bump or a piece of dirt, then the line found by the least squares method will be off slightly from the true edge. Therefore, in Step 130, the RMS error found by the least squares method is tested. If the RMS error is less than .5 pixels, then the edge data is assumed to be good and no further processing is performed. If the RMS error is in an intermediate range (.5 to 1.5 pixels), then at Step 135 a verification percentage is calculated for the least squares line. The verification percentage is a calculation of what percentage of the edge points fall within a given distance (closer than 1.5 pixels) of the fitted line. If the verification percentage is more than 90%, then no further processing is required. Of course, the variables used to determine whether more processing is required (the high and low RMS error, the verification percentage and the verification distance) can be adjusted for best results for a particular application. If the two tests applied in Steps 130 and 135 are both failed, this indicates that there are some bad points in the edge point data which do not correspond to the true edge. The edge data is further analyzed to determine which edge points correspond to the true edge and which correspond to noise or imperfections.

The selection of "true" (substantially collinear) edge points is performed by finding the slope and intercept of a line which has the maximum number of edge points on it. The line is then used as a basis for selecting the points to be used to fit the final line.

Referring to Fig. 2B, the details of this selection method are shown. At Step 140, an initial guess is made at the slope of the line. In many applications, such as registration of layers of semiconductors, it is known in advance what the expected slope of the edge is. This expected slope will be used as the first trial slope. First, however, the edge data

is converted to integer x-y pixel values to speed up the calculation. An n x 4 array ("EDGE") is created, where n is the number of edge points. The columns of the EDGE array will store the x pixel value, the y pixel value, a y-intercept value and a flag.

At Step 150, the trial slope is used to calculate, for each point, the y-intercept of a line having the trial slope and passing through the point. By describing the line in the standard slope, intercept fashion:

$$y = mx + b$$

then the y-intercept of a line passing through the point x,y and having slope m can be calculated as:

$$b = y - mx$$

The resulting y-intercept is stored in the EDGE array.

A histogram is constructed of the y-intercepts of lines passing through each edge point and having slope equal to the trial slope. The y-intercept values are quantized into "bins" one pixel wide, and each bin contains the number of edge points on the corresponding line. The count of edge points in the histogram bin containing the most edge points is selected as the peak value for this trial slope.

At Step 160, a new trial slope is selected. The angle (θ) of the expected slope is calculated using the inverse tangent function. The angle is incremented in a first direction by a predetermined amount (to the left by .5 degrees, in this embodiment). The tangent of the new angle is calculated to determine the next trial slope. The y-intercepts of the lines having the new slope and passing through each point are calculated, a

histogram is constructed, and the bin with the most points is determined. If the maximum number of points is greater than the previous maximum, then the search continues in the same direction, with the slope being incremented by an amount corresponding to .5 degrees in that direction. The search continues in the first direction until either: a) the angle reaches a predetermined limit; or b) the maximum number of pixels found in a bin has decreased for n consecutive tries (n = 4 in this embodiment). The search then proceeds in the other direction until one of these conditions is met. At Step 170, the trial slope corresponding to the histogram with the highest peak is selected as the best slope.

A subroutine for performing the search for the best slope (i.e. the slope of the line with the most pixels on it) is given in Appendix A as the function slope_search. The variables ANG_INC, ANG_MAX and LIMIT_TEST determine the angular increment for the search, the maximum angle from the expected slope angle, and the number of times the result must decrease for the search to stop, respectively. These parameters may be varied for the best results in a particular application.

It will be noted that in the subroutine slope_search, the start_ang variable is calculated once from the expected slope using the arctangent function. The slope for each new test is calculated once for each test using start_ang, an increment, and the tangent function before the call to subroutine match_slope. The invention thus increases computational efficiency by minimizing the use of transcendental functions. This is in contrast to the prior art methods which use angle-radius coordinates, requiring repeated calculation of trigonometric functions. Such methods were developed in response to earlier methods which used the slope

intercept equation, where the slope is unbounded and does not provide a good independent variable for the search. The preferred embodiment of the present invention uses the angle of the slope as an independent variable, computes the slope using the tangent function, and computes the y-intercept value without requiring further calculation of transcendental functions.

For each trial slope value, subroutine slope_search calls subroutine match_slope to accumulate the histogram for that slope. Subroutine match_slope computes the y-intercepts and calls subroutine histogram once for each value to enter the value into the histogram. Subroutine match_slope then finds the peak bin in the histogram for the trial slope and returns to subroutine slope_search for calculation of the next trial slope. Subroutine slope_search determines the peak value from all histograms and the slope corresponding to this peak value. Subroutine slope_search then makes one final call to subroutine match_slope to rebuild the histogram for the best slope.

After the search is complete, the "peak" bin has been found, and the best slope has been determined, the histogram for that slope is further examined (Step 175). For a given slope, all edge points whose y-intercept values are identical will lie on the same line. In practice, however, edge points that lie along the same line will not necessarily have identical y-intercepts, although they will be very close. It is likely that some of the good edge points will, because of noise, fall into bins immediately adjacent to the peak bin. Therefore, on the last call to subroutine match_slope, the peak bin and adjacent bins are examined and good edge points are selected and flagged. Starting with the peak bin of the histogram, either the left or right adjacent bin, whichever is higher, is added to the

selected points. This process continues until either: a) the total number of points selected exceeds a predetermined percentage (65% in this embodiment) of the total number of edge points; or b) the highest adjacent bin has less than 10% of the number of total edge points. In the latter case, the bin with less than 10% of the total is not selected.

The selected points are flagged in the fourth column of the EDGE array. Preferably, an additional scan is made through the edge data and, if any point is not selected, than the adjacent points on both sides of that point are also marked as "bad". This takes care of the common case where edge data is good for a portion of the image and is then corrupted by dirt. The last edge point along the good portion of the line may be perturbed by the presence of dirt, but its data may still be good enough to have been selected. This final scan will throw out all such points that are adjacent to noisy areas, avoiding a small perturbation in the results.

At Step 180, a final least squares line fit is performed, using only the selected points. It will be noted that although the selection of good edge points was performed with integer arithmetic, allowing the program to execute faster, the final line fitting calculation is now performed with floating point edge data.

At Step 190, a verification percentage is calculated for the line obtained by the least squares fit. The verification percentage determines if the method was able to extract a valid line from the data. (It is possible, of course, that there is no valid line.) The verification percentage is a calculation of what percentage of the edge points fall within a given distance (1.5 pixels in this embodiment) of the fitted line. If the percentage falls below a limit (which may

be set by the user), then the data is rejected as too noisy.

After the equations for the lines approximating the edges have been determined, the distance between the lines can be measured. If the slopes are not equal, the distance can be measured at the center point on each line. The resulting measurement can be displayed, directly or indirectly, on display screen 26 (Fig. 1). A number may be displayed, or a graphical indication of the measurement may be displayed. In the semiconductor mask alignment application, several sites are visited on a wafer, measurements are performed, and a vector map is displayed, showing areas that are not properly registered, with a graphical indication of the amount of the error.

Figs. 3-4 illustrate some typical results of the above-described method in operation. Fig. 3 illustrates the grey scale scanlines generated from a camera scan of a straight edge with a bump. Fig. 4A illustrates the histogram 40 resulting from calculation of y-intercepts of lines passing through the edge data at the expected vertical slope 42. The bump in the material shows up as a small peak in the histogram at this slope. Fig. 4B illustrates the histogram 44 generated at a typical search slope 46. Fig. 4C illustrates the histogram 48 generated when the search slope 50 corresponds to the actual slope of the edge. A very prominent peak in the histogram is generated when the actual edge slope is used. The bump is reflected in the histogram by small side lobes. The range of angles chosen for search slopes in Figs. 4A-4C has been exaggerated for purposes of illustration.

A preferred embodiment of the invention has now been described. It will be understood, however, that many variations and modifications will be apparent to

those of ordinary skill of the art informed by this disclosure. Further, there are many applications where the line detection aspects of the invention are useful even though a separation measurement is not required. All such applications, variations and modifications are well within the scope of the present invention, which is defined by the following claims.

```
#include "map-c.h"

#define LEFT       0
#define RIGHT      1

#define PI         3.14159265358979323846
#define DEG2RAD    (PI/180.0)

#define ANG_INC    (0.50*DEG2RAD)
#define ANG_MAX    (10.0*DEG2RAD)
#define LIMIT_TEST  4

#define X_CENTER  376
#define Y_CENTER  240

#define BAD_Y_INT 0x4              /* bad y_intercept */


/* VERIFY_DISTANCE is the distance in pixels that an edge point
can be
   from the calculated edge to count as a verification point */

#define VERIFY_DISTANCE  1.5

#define TIGHT_RMS        0.5
#define LOOSE_RMS        1.5
#define TIGHT_VFY        0.90
#define LOOSE_VFY         0.10                /* set way low for
experimentation */

/*  When match_slope() analyzes the histogram to mark edge points
for
    selection, it includes the max bin plus enough adjacent side
bins
    so that SELECT_PCT of the edge points are selected.  If some
almost
    empty bins are found before SELECTION_PCT is reached, then
the
    selection process terminates immediately.   This avoids
including
    noise points in a very noisy picture. */

#define SELECTION_PCT   0.65  /* selection % expressed as a
fraction */

/* EXTERNALS */

extern struct sys_rec_type    sys_rec;
extern struct meas_rec_type   *meas_ptr;


/* GLOBALS for this source file */

struct hnode {
    struct hnode *left;
    struct hnode *right;
```

```
    int             value;
    int             count;
};

static struct hnode far *hptr;
static struct hnode far *hhead;
static struct hnode far *htail;
static struct hnode far *hist_alloc;
static struct hnode far *hist_free;

static double x_pix_siz;
static double y_pix_siz;

static int horiz_wind;


/*****************************************************************
********

 fit_line() -- main entry point for new line fitting algorithms

 inputs:  coords[][2]    -- floating point edge data
          count          -- length of coords[][2]

 outputs: slope          -- slope of line fit to edge points
          y_intercept    -- y_intercept of line fit to edge points


  return value:    -1          out of memory
                    0          normal return, line found
                    1          line found, but loose verify only
                    2          line not reliably found

*****************************************************************
********/
int fit_line(coords, count, slope, y_intercept)
double coords[][2],
       *slope,
       *y_intercept;
int    *count;
{
    void far *edge_alloc;
    unsigned int seg, ofs;
    unsigned long long_addr;
    int far (*edge)[4];
    double expected_slope, rms, vfy_pct;
    int rc;
    char out_string[80];

    x_pix_siz = sys_rec.x_pixel[meas_ptr->obj_num];
    y_pix_siz = sys_rec.y_pixel[meas_ptr->obj_num];

    horiz_wind =
         ( (meas_ptr->is_window && meas_ptr->meas_edge.wind.type
== HORIZ_WIND)
       ||   (!meas_ptr->is_window && meas_ptr->gate[0].type ==
```

```
HORIZ_GATE) );

    /* set edge to dummy value to keep compiler from issuing
warning message */
    edge = NULL;

    least_sqr(coords, *count, slope, y_intercept, edge, &rms,
FALSE);
    if(!PURGE_ON)
        return(0);

    if(rms <= TIGHT_RMS) {
/*
        DEBUG3("FIT_LINE: quick return, rms = %.2f, m,b = %8.5f
%8.3f",
                                                rms,    *slope,
*y_intercept);
*/
        return(0);
    }

    if(rms <= LOOSE_RMS) {
        verify_line(coords,   *count,   *slope,   *y_intercept,
&vfy_pct);
        if(vfy_pct >= TIGHT_VFY) {
/*
            DEBUG4("FIT_LINE: quick verify, rms = %.2f  vfy =
%.2f, m,b = %8.5f %8.3f",
                                            rms, vfy_pct, *slope,
*y_intercept);
*/
            return(0);
        }
    }

    /* Normalize the pointers returned from malloc.   (They
probably
       are already, but better safe than sorry.)  Since we know
that
       the "edge" and "hist" arrays are less than 64k, we can
safely
       use short arithmetic on pointers into these structures.
(I.e.,
       we can declare pointers to them "far".)  */

    edge_alloc = malloc(4*sizeof(int)*MAX_COORDS);
    if(edge_alloc == NULL) {
        display_error("FIT_LINE: Not enough memory. Unable to fit
line.");
        return(-1);
    }
    seg = FP_SEG(edge_alloc);
    ofs = FP_OFF(edge_alloc);
    long_addr = (((unsigned long)seg) << 4) + ofs;
    seg = (unsigned int)(long_addr >> 4);
    ofs = (unsigned int)(long_addr & 0xF);
```

```
    edge = (int far (*)[4])(((unsigned long)seg<<16) | ofs);

    hist_alloc = malloc(sizeof(struct hnode)*MAX_COORDS);
    if(hist_alloc == NULL) {
        display_error("FIT_LINE: Not enough memory. Unable to fit
line.");
        free((void *)edge_alloc);
        return(-1);
    }
    seg = FP_SEG(hist_alloc);
    ofs = FP_OFF(hist_alloc);
    long_addr = (((unsigned long)seg) << 4) + ofs;
    seg = (unsigned int)(long_addr >> 4);
    ofs = (unsigned int)(long_addr & 0xF);
    hist_alloc = (struct hnode far *)(((unsigned long)seg<<16)
| ofs);

    quantize_edge(coords, *count, edge);

    expected_slope= 0.0;
    slope_search(coords, *count, expected_slope, edge);

    least_sqr(coords, *count, slope, y_intercept, edge, &rms,
TRUE);

    verify_line(coords, *count, *slope, *y_intercept, &vfy_pct);


    free((void *)edge_alloc);
    free((void *)hist_alloc);

    if(rms <= LOOSE_RMS) {
        if(vfy_pct >= TIGHT_VFY)
            rc = 0;
        else if(vfy_pct >= LOOSE_VFY)
            rc = 0;                          /* set rc to 0 for now
*/
        else
            rc = 2;
    } else {
        rc = 2;
    }

/*
    DEBUG4("FIT_LINE: fitted line, rms = %.2f  vfy = %.2f, m,b
= %8.5f %8.3f",
                                             rms, vfy_pct, *slope,
*y_intercept);
*/
    return(rc);
}

/*****************************************************************
********

 quantize_edge() -- Calculate the integer X, Y pixel values of
```

```
each
        edge point and stuff it in edge array.

 inputs:  coords[][2]   -- floating point edge data
          count         -- length of coords[][2] (and edge[][4])

          edge[][4]     -- dynamically allocated companion array
                                for edge data

 outputs: edge[i][0]    -- X pixel value for coords[i][0]
          edge[i][1]    -- Y pixel value for coords[i][1]
          edge[i][2]    -- y-intercept for edge point i
          edge[i][3]    -- flag word for edge point i

**********************************************************************
********/
quantize_edge(coords, count, edge)
double coords[][2];
int    count;
int    edge[][4];
{
    int i;
    double x_start, y_start;
    double inv_x_pix_siz, inv_y_pix_siz;

    x_start = meas_ptr->meas_edge.wind.x_start;
    y_start = meas_ptr->meas_edge.wind.y_start;
    inv_x_pix_siz = 1.0/x_pix_siz;
    inv_y_pix_siz = 1.0/y_pix_siz;


    /* if horizontal window, X and Y have been swapped */
    if(horiz_wind) {
        for(i=0; i<count; i++) {
                edge[i][0]   =   -floor(0.5   +
coords[i][0]*inv_y_pix_siz);
                edge[i][1]   =       floor(0.5   +
coords[i][1]*inv_x_pix_siz);
            edge[i][0] += Y_CENTER - y_start;
            edge[i][1] += X_CENTER - x_start;
        }
    } else {
        for(i=0; i<count; i++) {
                edge[i][0]   =       floor(0.5   +
coords[i][0]*inv_x_pix_siz);
                edge[i][1]   =   -floor(0.5   +
coords[i][1]*inv_y_pix_siz);
            edge[i][0] += X_CENTER - x_start;
            edge[i][1] += Y_CENTER - y_start;
        }
    }

    return(0);
}
```

```
/*****************************************************************
********

 calc_fitsums() -- calculate sums and sums of products needed for
least
        square line fitting.

 This routine assumes that coords[i][0] is the X value, and
coords[i][1]
 is the Y value.  In fact, the coordinates may have been flipped
prior to
 the call, but it is up to the calling routine to account for
that.
*****************************************************************
*********/
calc_fitsums_new(count, coords, edge, x_sum, y_sum, x2_sum,
y2_sum,
                                                    xy_sum,     n,
select)
int count, *n, select;
int edge[][4];
double coords[][2];
double *x_sum,
       *y_sum,
       *x2_sum,
       *y2_sum,
       *xy_sum;
{
    int i;
    double x, y;

    *n = 0;
    *x_sum = *y_sum = *x2_sum= *y2_sum = *xy_sum = 0;
    if(select) {
        for (i=0; i<count; i++) {
            if (!edge[i][3]) {
                *x_sum += (x=coords[i][0]);
                *x2_sum += x*x;
                *y_sum += (y=coords[i][1]);
                *y2_sum += y*y;
                *xy_sum += x*y;
                (*n)++;
            }
        }
    } else {
        for (i=0; i<count; i++) {
            *x_sum += (x=coords[i][0]);
            *x2_sum += x*x;
            *y_sum += (y=coords[i][1]);
            *y2_sum += y*y;
            *xy_sum += x*y;
            (*n)++;
        }
    }
```

```
    return(0);
}

/*****************************************************************
********

 least_sqr() -- routine to perform a least squares line fit to
the
        qualified edge points.


This routines assumes that coords[i][0] is the X value and
coords[i][1]
is the Y value, and that it is fitting a horizontal line to the
data.  In
fact, the X and Y data could have been flipped if it came from
a horizontal
window.  It is up to the calling program to account for that.

CAUTION:  This program will blow up if the line is oriented the
wrong way.
        A least squares fit based on y=mx+b will try to minimize
the errors
        in y of the fitted line and this will become wildly
erratic if the
        line approaches the vertical.

        For now, I've flipped the algorithm to base it on x=my+b

*****************************************************************
********/
least_sqr(coords, count, slope, y_intercept, edge, rms, select)
double coords[][2],
       *slope,
       *y_intercept,
       *rms;
int    count,
       select;
int    edge[][4];
{
    double x_sum,
           y_sum,
           x2_sum,
           y2_sum,
           xy_sum,
           temp,
           dev_sq;
    int    n;


    calc_fitsums_new(count,  coords,  edge,  &x_sum,  &y_sum,
&x2_sum, &y2_sum, &xy_sum, &n, select);

    /* We should never see temp=0.  About the only way for this
to occur is
```

```
        if all of the points are thrown out and n=0.  (This should
not happen
        either.)  In any event, a little defensive programming
seems
        worthwhile. */

    temp = n * x2_sum - x_sum * x_sum;
    if (temp==0.0 || n==0) {
        *slope = 0;
        *y_intercept = 0;
        *rms = 1000;            /* make sure line is thrown out
*/
    } else {
        *slope = (n * xy_sum - x_sum * y_sum)/temp;
        *y_intercept = (x2_sum * y_sum - x_sum * xy_sum)/temp;
        dev_sq = y2_sum - *y_intercept * y_sum - *slope * xy_sum;

        *rms = sqrt(dev_sq/n);
    }

    if(horiz_wind)
        *rms = *rms/x_pix_siz;
    else
        *rms = *rms/y_pix_siz;

    return(0);
}


/*****************************************************************
********

 verify_line -- calculate how many of the edge points meet the
        verification criterion.

*****************************************************************
********/
verify_line(coords, count, m, b, vfy_pct)
int count;
double coords[][2],
       m,
       b,
       *vfy_pct;
{
    double tolerance;
    int i, vfy_count;

    vfy_count = 0;

    if(horiz_wind)
        tolerance = x_pix_siz * VERIFY_DISTANCE;
    else
        tolerance = y_pix_siz * VERIFY_DISTANCE;

    for (i=0; i<count; i++) {
        if  (fabs((m  *  coords[i][0]  +  b)  -  coords[i][1])  <
```

```
tolerance)
            vfy_count++;
    }

    if(horiz_wind)
                                * v f y _ p c t =
(double)vfy_count/meas_ptr->meas_edge.wind.delta_y;
    else
                                * v f y _ p c t =
(double)vfy_count/meas_ptr->meas_edge.wind.delta_x;

    return(0);
}


/*****************************************************************
********

 histogram() -- accumulates histogram data.  Called once for each
value
        entered into the histogram.

 inputs:  value -- the value to be accumulated

*****************************************************************
********/
histogram(value)
int value;
{
    struct hnode *hnew, *hprev;


    if(value == hptr->value) {
        hptr->count++;
    } else if (value < hptr->value) {

        for(;;) {                                   /* scan left */
            if(hptr->left == NULL) {
                hnew = (struct hnode far *)hist_free++;
                hptr->left = hnew;
                hnew->left = NULL;
                hnew->right = hptr;     /* add node at left end
*/
                hnew->value = value;
                hnew->count = 1;
                hptr = hhead = hnew;
                break;
            } else {
                hptr = hptr->left;
                if(value == hptr->value){
                    hptr->count++;
                    break;
                } else if(value < hptr->value) {
                    continue;
                } else {
```

```
                    hnew = (struct hnode far *)hist_free++;
                    hprev = hptr->right;
                    hptr->right = hprev->left = hnew;
                    hnew->left = hptr;
                    hnew->right = hprev;
                    hnew->value = value;
                    hnew->count = 1;
                    hptr = hnew;
                    break;
                }
            }
        }

    } else {

        for(;;) {                                   /* scan right */

            if(hptr->right == NULL) {
                hnew = (struct hnode far *)hist_free++;
                hptr->right = hnew;
                hnew->right = NULL;
                hnew->left = hptr;      /* add node at right end
*/
                hnew->value = value;
                hnew->count = 1;
                hptr = htail = hnew;
                break;
            } else {
                hptr = hptr->right;
                if(value == hptr->value){
                    hptr->count++;
                    break;
                } else if(value > hptr->value) {
                    continue;
                } else {
                    hnew = (struct hnode far *)hist_free++;
                    hprev = hptr->left;
                    hptr->left = hprev->right = hnew;
                    hnew->left = hprev;
                    hnew->right = hptr;
                    hnew->value = value;
                    hnew->count = 1;
                    hptr = hnew;
                    break;
                }
            }
        }
    }

    return(0);
}

/*****************************************************************
*******
```

```
 match_slope() -- fits a test line of the specified slope through
each
        of the edge points and accumulates a histogram of the
y_intercepts
        of the lines.  If the line in the image has the same
slope as the
        test line, then the histogram will exhibit a large peak
marking
        the location of the line.  This routine returns the value
(in
        number of pixels) of the highest peak in the current
histogram.

 inputs:  coords[][2]   -- raw edge data
          count        -- count of edge points in coords[][] and
edge[][]
          slope         -- slope of test line
          edge[][4]     -- processed edge data and flags
          mark          -- tells whether to mark edge[][] flags
(T/F)

 outputs: num                -- number of pixel in max bin of
histogram

**********************************************************************
********/
match_slope(coords, count, slope, edge, num, mark)
double coords[][2],
       slope;
int    count, *num;
int    edge[][4];
{
    double pix_slope;

    int (*e_ptr1)[4], (*e_ptr2)[4];    /* pointers for edge[][4]
array */

    int i, lower_limit, upper_limit, max;
    int bin_count, right_count, left_count;

    struct hnode *hmax, *r_hptr, *l_hptr;


    /*initialization for histogram */
    hist_free = hist_alloc;
    hptr = hhead = htail = (struct hnode far *)hist_free++;
    hptr->right = hptr->left = NULL;
    hptr->value = 0;                        /* these two create a
null node */
    hptr->count = 0;                        /* may want to handle
differently */


/**********************************************************************
```

```
     algorithm to compute y_int[] array using edge[][4] data.
This
       requires that the edge data be calculated, but then is
quicker
       for subsequent passes.

       (Note:  Coding this loop in assembly language will
substantially
               speed up the line searching algorithm.  The
quickest
               way to do it would be to set the 80387 for round
to
               nearest mode outside the loop, execute the loop,
and
               reset the 80387 to its previous state.  This will
be
               MUCH faster than adding 0.5 and calling floor().
)

       Note that we must invert the sign of slope since the
positive Y
       sense flips when we go from stage (micron) coordinates to
pixel
       coordinates.

*****************************************************************/


    /* compute y_int for each edge point and quantize it */
    if(horiz_wind) {
        pix_slope = -slope * (y_pix_siz/x_pix_siz);
    } else {
        pix_slope = -slope * (x_pix_siz/y_pix_siz);
    }
    e_ptr2 = edge + count;
    for (e_ptr1=edge; e_ptr1<e_ptr2; e_ptr1++) {
        histogram( (*e_ptr1)[2] =
           floor(0.5 + (*e_ptr1)[1] - pix_slope * (*e_ptr1)[0])
);
    }

    /* find bin with largest number of edge points */
    max = 0;
    for (hptr=hhead; hptr!=NULL; hptr=hptr->right) {
        if (hptr->count > max) {
            max = hptr->count;
            hmax = hptr;
        }
    }

    *num = max; /* return to calling program */


  /*****************************************************************

    Updated bin selection:
```

```
    Select max bin plus enough adjacent bins to include at least
    SELECTION_PCT of edge points.
***********************************************************************/

    if(mark) {
        bin_count = max;
        r_hptr = l_hptr = hmax;
        lower_limit = upper_limit = hmax->value;
        for (;;) {
            if (bin_count > SELECTION_PCT * count)
                break;

            if (l_hptr->left != NULL)
                left_count = l_hptr->left->count;
            else
                left_count = 0;

            if (r_hptr->right != NULL)
                right_count = r_hptr->right->count;
            else
                right_count = 0;


            if (right_count > left_count) {
                if (right_count < 0.1 * count)
                    break;
                r_hptr = r_hptr->right;
                upper_limit = r_hptr->value;
                bin_count += r_hptr->count;

            } else {
                if (left_count < 0.1 * count)
                    break;
                l_hptr = l_hptr->left;
                lower_limit = l_hptr->value;
                bin_count += l_hptr->count;
            }
        }

        /* initialize all points to good */
        for (i=0; i<count; i++)
            edge[i][3] = 0;

        /* mark bad edge points */
        for (i=0; i<count; i++) {
            if(edge[i][2]  <  lower_limit  ||  edge[i][2]  >
upper_limit) {
                edge[i][3] |= BAD_Y_INT;
                if (i > 0)
                    edge[i-1][3] |= BAD_Y_INT;
                if (i < count-1)
                    edge[i+1][3] |= BAD_Y_INT;
```

```
            }
        }
    }

    return(0);
}


/*****************************************************************
********

 slope_search() -- conduct search for a line using match_slope
routine.

 inputs:  coords[][2]     -- raw edge data
          count           -- count of edge points in coords[][]
and edge[][]
          expected_slope -- expected slope of line
          edge[][4]       -- processed edge data and flags

*****************************************************************
********/
slope_search(coords, count, expected_slope, edge)
double coords[][2],
       expected_slope;
int    count;
int    edge[][4];
{
    int done[2];
    double start_ang, slope;
    int dir;
    int num, max, i_max[2], limit[2];


    start_ang = atan(expected_slope);

    match_slope(coords,  count,  expected_slope,  edge,  &num,
FALSE);
    max = num;

    /* arbitrarily start search to the left */
    slope = tan(start_ang - ANG_INC);
    match_slope(coords, count, slope, edge, &num, FALSE);
    if(num > max) {
        max = num;
        i_max[LEFT] = i_max[RIGHT] = -1;
        dir = LEFT;
    } else if (num < max) {
        i_max[LEFT] = i_max[RIGHT] = 0;
        dir = RIGHT;
    } else {
        i_max[LEFT] = -1;                /* num == max */
        i_max[RIGHT] = 0;
        dir = LEFT;
    }
```

```
limit[LEFT]  = -1;  /* limit left */
limit[RIGHT] =  0;  /* limit right */

dir = LEFT;
done[LEFT] = done[RIGHT] = FALSE;

while (!done[LEFT] || !done[RIGHT]) {

  switch (dir) {
    case LEFT:
        if(done[LEFT]) continue;

        if(limit[LEFT] * ANG_INC <= -ANG_MAX) {
            done[LEFT] = TRUE;
            dir = RIGHT;
            continue;
        } else {
            limit[LEFT] -= 1;
        }

        slope = tan(start_ang + limit[LEFT] * ANG_INC);
        match_slope(coords, count, slope, edge, &num, FALSE);

        if(num > max) {
            max = num;
            i_max[LEFT] = i_max[RIGHT] = limit[LEFT];
        } else if (num == max) {
            i_max[LEFT] = limit[LEFT];
        }

        if(abs(i_max[LEFT] - limit[LEFT]) > LIMIT_TEST) {
            done[LEFT] = TRUE;
            dir = RIGHT;
        }
        break;

    case RIGHT:
        if(done[RIGHT]) continue;

        if(limit[RIGHT] * ANG_INC >= ANG_MAX) {
            done[RIGHT] = TRUE;
            dir = LEFT;
            continue;
        } else {
            limit[RIGHT] += 1;
        }

        slope = tan(start_ang + limit[RIGHT] * ANG_INC);
        match_slope(coords, count, slope, edge, &num, FALSE);

        if(num > max) {
            max = num;
            i_max[RIGHT] = i_max[LEFT] = limit[RIGHT];
        } else if (num == max) {
            i_max[RIGHT] = limit[RIGHT];
        }
```

```
            if(abs(i_max[RIGHT] - limit[RIGHT]) > LIMIT_TEST) {
                done[RIGHT] = TRUE;
                dir = LEFT;
            }
            break;
        }

    } /*** end of while (!done[LEFT] || !done[RIGHT]) ***/

    /* call match_slope one last time to mark edge[][] flags */
    slope = tan(start_ang + i_max[LEFT] * ANG_INC);

    match_slope(coords, count, slope, edge, &num, TRUE);

    return(0);
}

^Z
```

WHAT IS CLAIMED IS:

1. A method of detecting collinear edge points in a digitized image comprising the steps of:

a) capturing an image of an object;

b) digitizing the captured image to generate an ordered set of pixel values;

c) identifying a set of edge points from the ordered set of pixel values;

d) setting a trial slope equal to an expected slope value;

e) for each one of said identified edge points, computing an intercept value of a straight line passing through the identified edge point and having a slope equal to the trial slope;

f) accumulating a count of identified edge points associated with the computed intercept value and the trial slope;

g) varying the expected slope by an amount corresponding to a predetermined angular increment to generate a new trial slope;

h) repeating steps e) and f) for the new trial slope to generate a plurality of accumulated counts, each accumulated count being associated with a slope/intercept pair;

i) comparing the accumulated counts of identified edge points to identify a best slope/intercept pair associated with the maximum count of identified edge points; and

j) selecting a collinear subset of the set of edge points by determining which ones of the identified edge points are on a straight line defined by the best slope/intercept pair.

2. The method of claim 1 wherein step g) of varying the expected slope comprises:

determining a start-angle associated with the expected slope;

incrementing the start-angle by a predetermined increment; and

calculating the tangent of the incremented start-angle.

3. The method of claim 2 further comprising the step of repeating steps g) and h) until a predetermined test is satisfied.

4. The method of claim 3 wherein step k) comprises generating a line by finding a least squares line fit for the selected subset of points.

5. The method of claim 3 wherein the step e) of computing an intercept value comprises the step of computing a y-intercept value.

6. The method of claim 3 wherein:

step i) comprises the step of identifying a best slope/intercept pair having a best slope and a best intercept; and

step j) comprises the steps of:

calculating, for each of said identified edge points, the intercept value of a straight line having a slope equal to the best slope and passing through said identified edge point; and

comparing each of the calculated intercept values to a range of values including the best intercept.

7. A method of measuring the distance between two edges on an object comprising the steps of:

a) capturing an image of the object;

b) digitizing the captured image to generate an ordered set of pixel values;

c) identifying a first set of edge points corresponding to a first edge from the ordered set of pixel values;

d) setting a trial slope equal to an expected slope value;

e) for each one of said identified edge points, computing an intercept value of a straight line passing through the identified edge point and having a slope equal to the trial slope;

f) accumulating a count of identified edge points associated with the computed intercept value and the trial slope;

g) varying the expected slope by an amount corresponding to a predetermined angular increment to generate a new trial slope;

h) repeating steps e) and f) for the new trial slope to generate a plurality of accumulated counts, each accumulated count being associated with a slope/intercept pair;

i) comparing the accumulated counts of identified edge points to identify a best slope/intercept pair associated with the maximum count of identified edge points;

j) selecting a collinear subset of the first set of edge points by determining which ones of the identified edge points are on a straight line defined by the best slope/intercept pair;

k) using the selected subset of the first set of edge points to generate a first line;

l) repeating steps a) through k) for a second set of edge points corresponding to a second edge to generate a second line; and

m) determining a distance between the first line and the second line.

8. The method of claim 7 wherein step g) of varying the expected slope comprises:

determining a start-angle associated with the expected slope;

incrementing the start-angle by a predetermined increment; and

calculating the tangent of the incremented start-angle.

9. The method of claim 8 further comprising the step of repeating steps g) and h) until a predetermined test is satisfied.

10. The method of claim 9 wherein step k) comprises generating a line by finding a least squares line fit for the selected subset of points.

11. The method of claim 9 wherein the step e) of computing an intercept value comprises the step of computing a y-intercept.

12. The method of claim 9 wherein:

step i) comprises the step of:

identifying a best slope/intercept pair having a best slope and a best intercept; and

step j) comprises the steps of:

calculating, for each of said identified edge points, the intercept value of a straight

line having a slope equal to the best slope and passing through said identified edge point; and

comparing each of the calculated intercept values to a range of values including the best intercept.

13. An apparatus for measuring the distance between two edges on an object comprising:

a camera for capturing an image of the object;

means coupled to said camera for digitizing the captured image to generate an ordered set of pixel values; and

image processing means coupled to said digitizing means for receiving and analyzing the ordered set of pixel values by:

a) identifying a first set of edge points corresponding to a first edge from the ordered set of pixel values;

b) setting a trial slope equal to an expected slope value;

c) for each one of said identified edge points, computing an intercept value of a straight line passing through the identified edge point and having a slope equal to the trial slope;

d) accumulating a count of identified edge points associated with the computed intercept value and the trial slope;

e) varying the expected slope by an amount corresponding to a predetermined angular increment to generate a new trial slope;

f) repeating steps c) and d) for the new trial slope to generate a plurality of accumulated counts, each accumulated count being associated with a slope/intercept pair;

g) comparing the accumulated counts of identified edge points to identify a best slope/intercept pair associated with the maximum count of identified edge points;

h) selecting a collinear subset of the first set of edge points by determining which ones of the identified edge points are on a straight line defined by the best slope/intercept pair;

i) using the selected subset of the first set of edge points to generate a first line;

j) repeating steps a) through i) for a second set of edge points corresponding to a second edge to generate a second line; and

k) determining a distance between the first line and the second line.

14. The apparatus of claim 13 wherein said image processing means includes means for analyzing the ordered set of pixel values by varying the expected slope by:

determining a start-angle associated with the expected slope;

incrementing the start-angle by a predetermined increment; and

calculating the tangent of the incremented start-angle.

15. The apparatus of claim 14 wherein said image processing means includes means for analyzing the ordered set of pixel values by varying the expected slope and comparing the accumulated counts until a predetermined test is satisfied.

16. The apparatus of claim 15 wherein said image processing means includes means for analyzing the

ordered set of pixel values by using the selected subset of points to generate a line by finding a least squares line fit for the selected subset of points.

17. The apparatus of claim 15 wherein said image processing means includes means for analyzing the ordered set of pixel values by:

comparing the accumulated counts to identify a best slope/intercept pair having a best slope and a best intercept; and

selecting a collinear subset by calculating, for each of said identified edge points, the intercept value of a straight line having a slope equal to the best slope and passing through said identified edge point and comparing each of the calculated intercept values to a range of values including the best intercept.

18. The apparatus of claim 15 wherein said image processing means comprises a programmed digital computer.

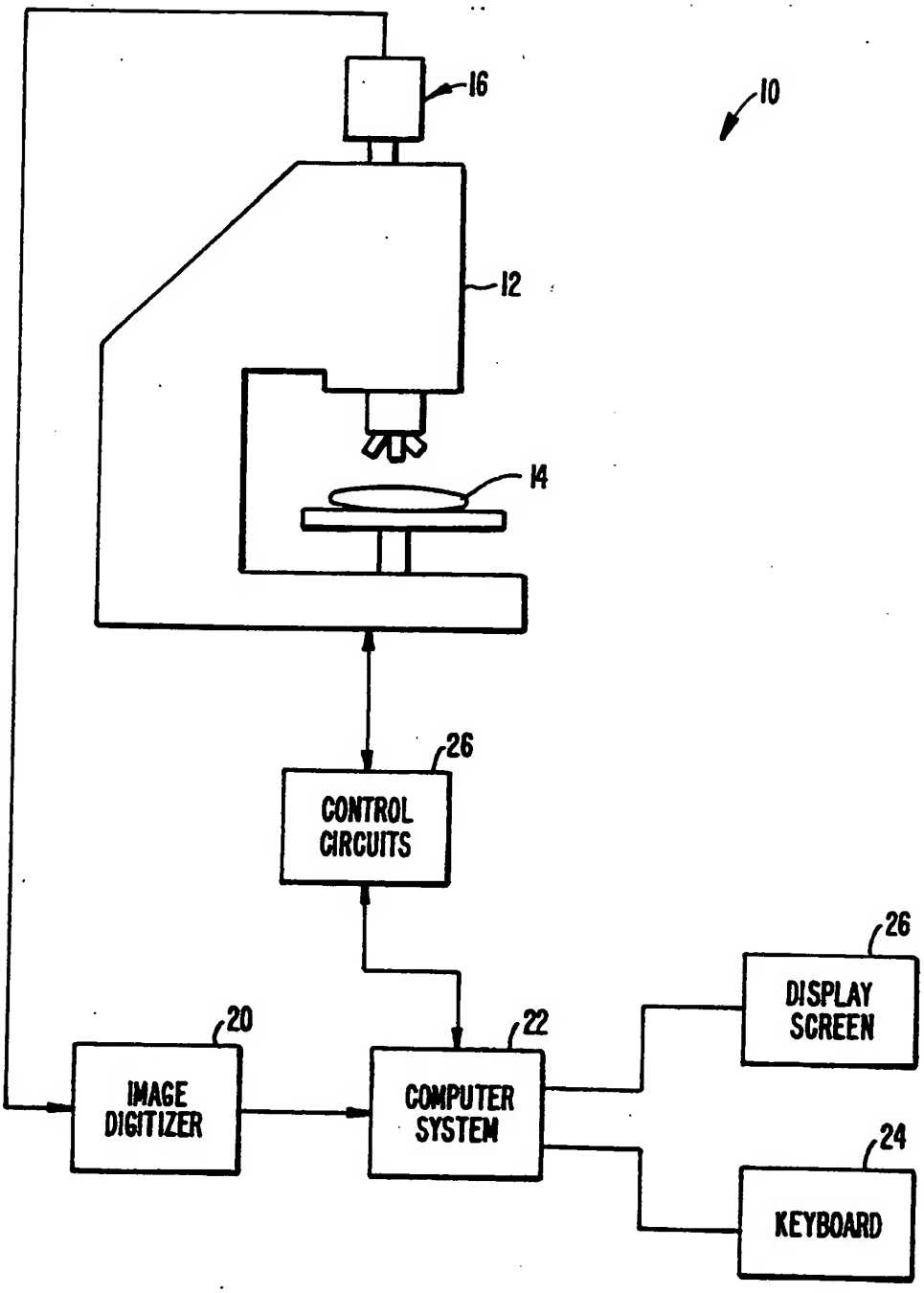

16
10
12
14
26
CONTROL CIRCUITS
26
DISPLAY SCREEN
20
IMAGE DIGITIZER
22
COMPUTER SYSTEM
24
KEYBOARD

FIG. 1.

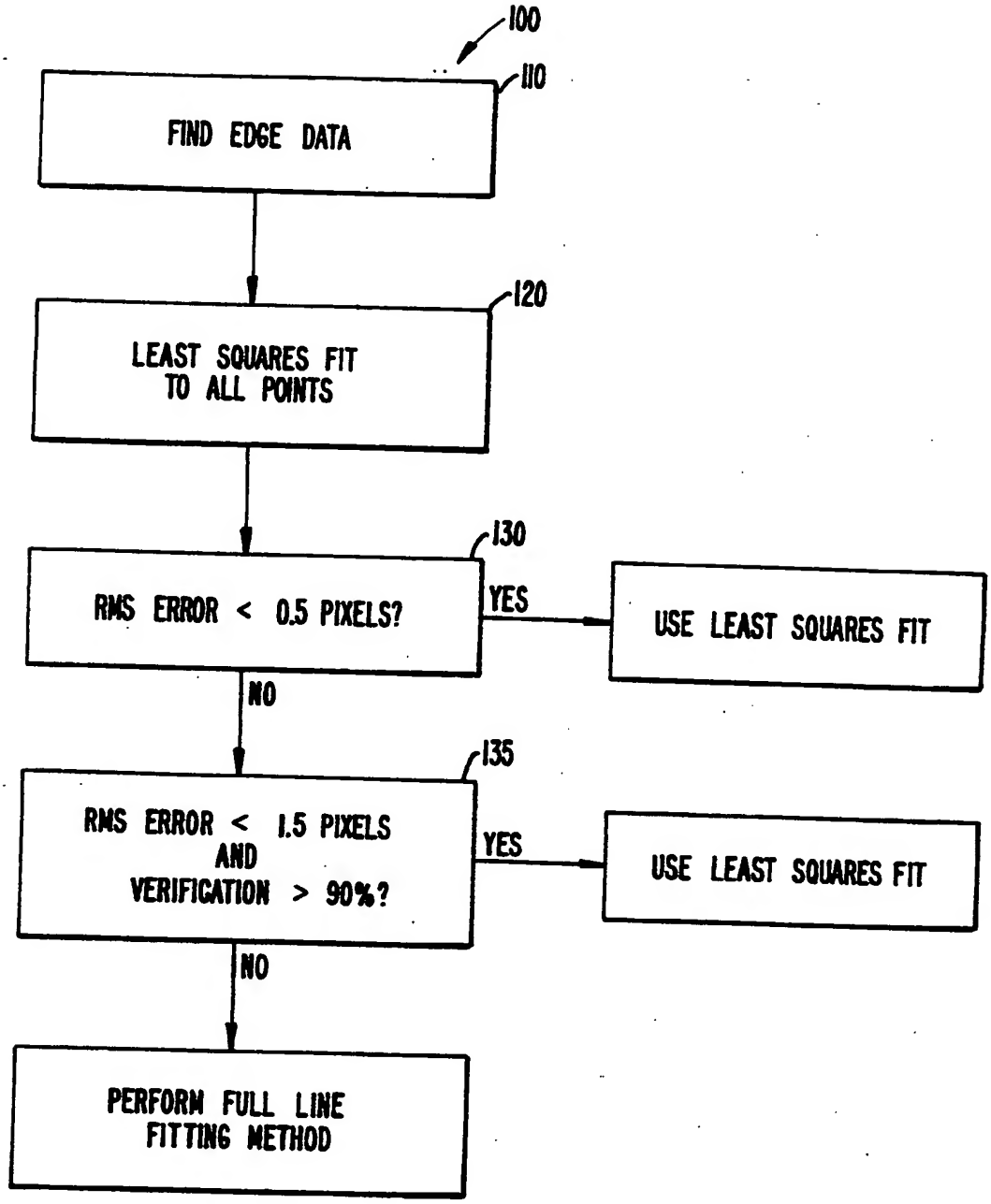

100
110
FIND EDGE DATA
120
LEAST SQUARES FIT TO ALL POINTS
130
RMS ERROR < 0.5 PIXELS?
YES
USE LEAST SQUARES FIT
NO
135
RMS ERROR < 1.5 PIXELS AND VERIFICATION > 90%?
YES
USE LEAST SQUARES FIT
NO
PERFORM FULL LINE FITTING METHOD

FIG. 2A.

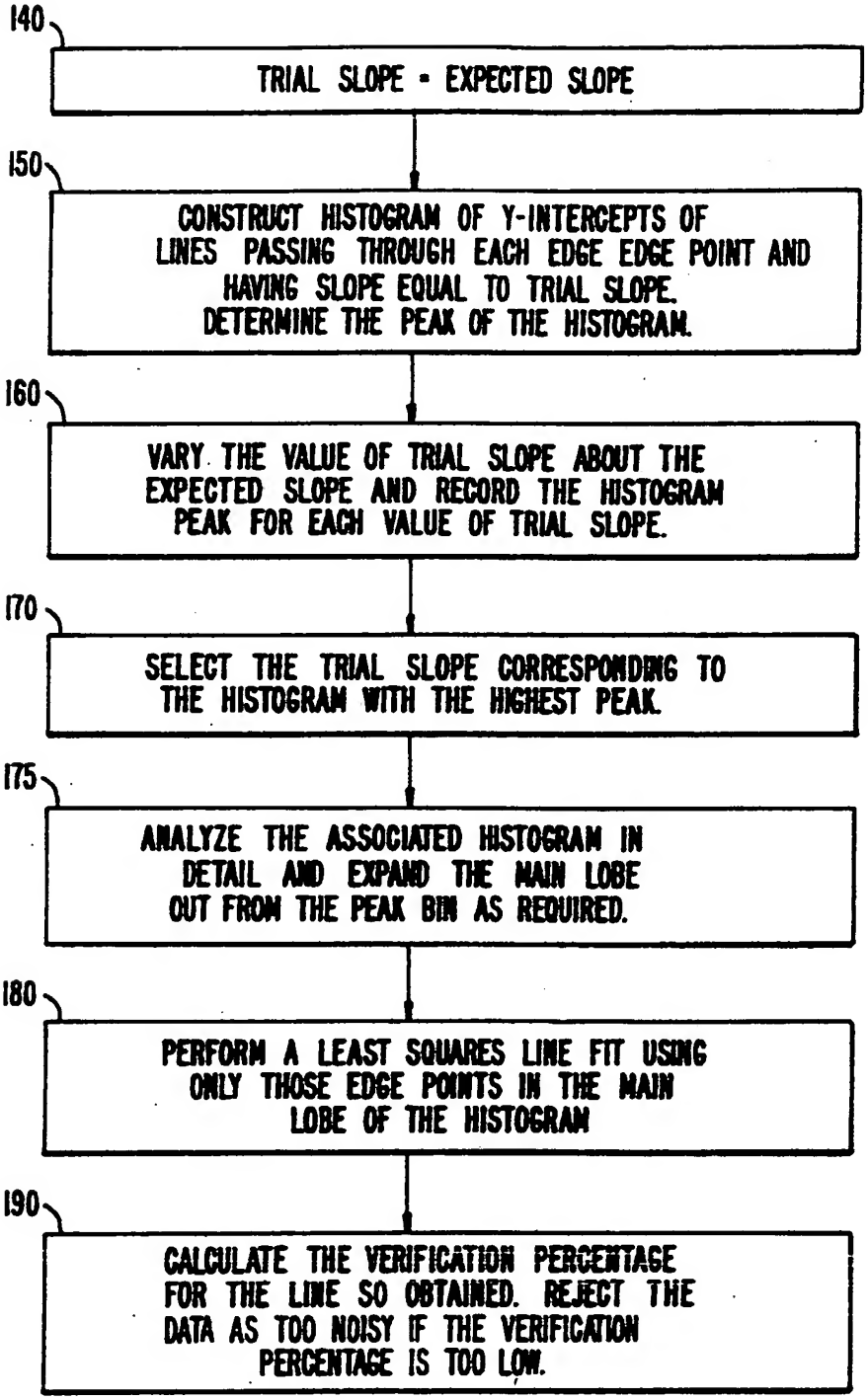
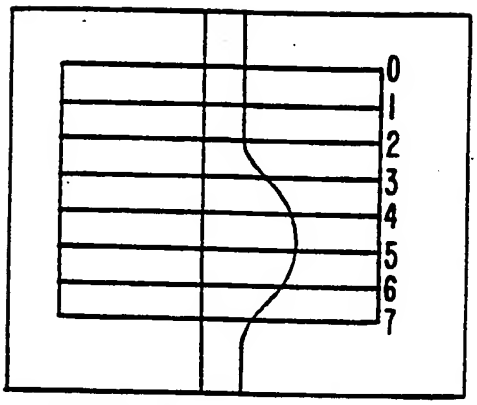

140
TRIAL SLOPE = EXPECTED SLOPE
150
CONSTRUCT HISTOGRAM OF Y-INTERCEPTS OF LINES PASSING THROUGH EACH EDGE EDGE POINT AND HAVING SLOPE EQUAL TO TRIAL SLOPE. DETERMINE THE PEAK OF THE HISTOGRAM.
160
VARY THE VALUE OF TRIAL SLOPE ABOUT THE EXPECTED SLOPE AND RECORD THE HISTOGRAM PEAK FOR EACH VALUE OF TRIAL SLOPE.
170
SELECT THE TRIAL SLOPE CORRESPONDING TO THE HISTOGRAM WITH THE HIGHEST PEAK.
175
ANALYZE THE ASSOCIATED HISTOGRAM IN DETAIL AND EXPAND THE MAIN LOBE OUT FROM THE PEAK BIN AS REQUIRED.
180
PERFORM A LEAST SQUARES LINE FIT USING ONLY THOSE EDGE POINTS IN THE MAIN LOBE OF THE HISTOGRAM
190
CALCULATE THE VERIFICATION PERCENTAGE FOR THE LINE SO OBTAINED. REJECT THE DATA AS TOO NOISY IF THE VERIFICATION PERCENTAGE IS TOO LOW.

FIG. 2B.



FIELD OF VIEW
SCANLINES 0-7 SHOWN
LENGTH 100 UNITS

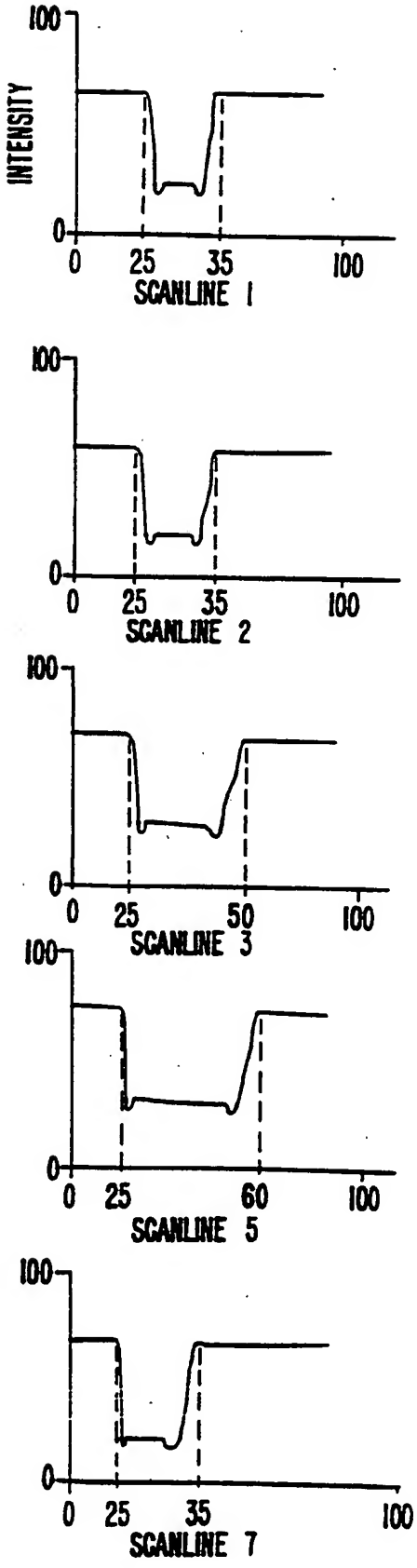


FIG. 3.

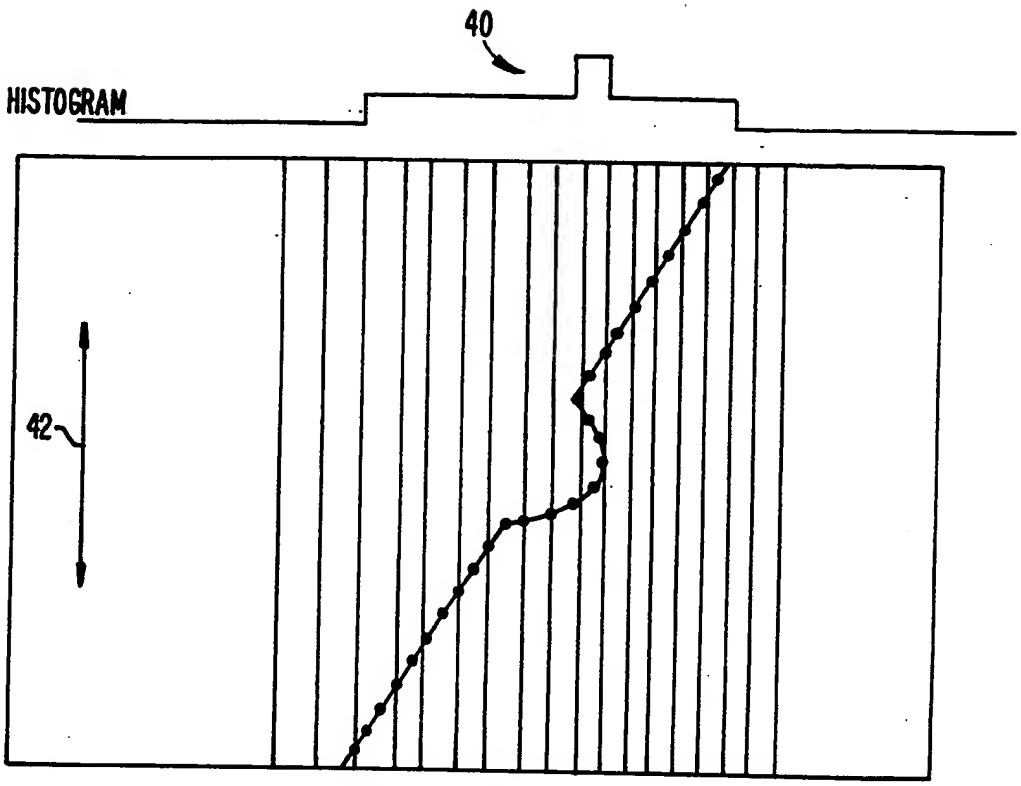


FIG. 4A.

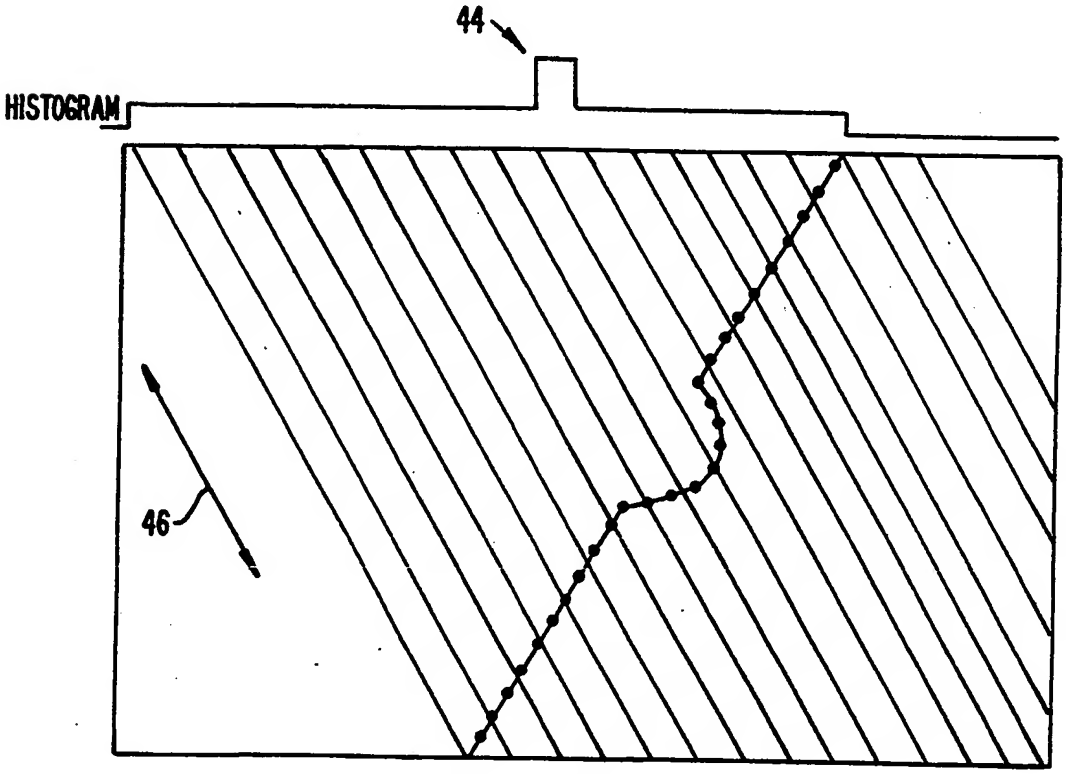


FIG. 4B.

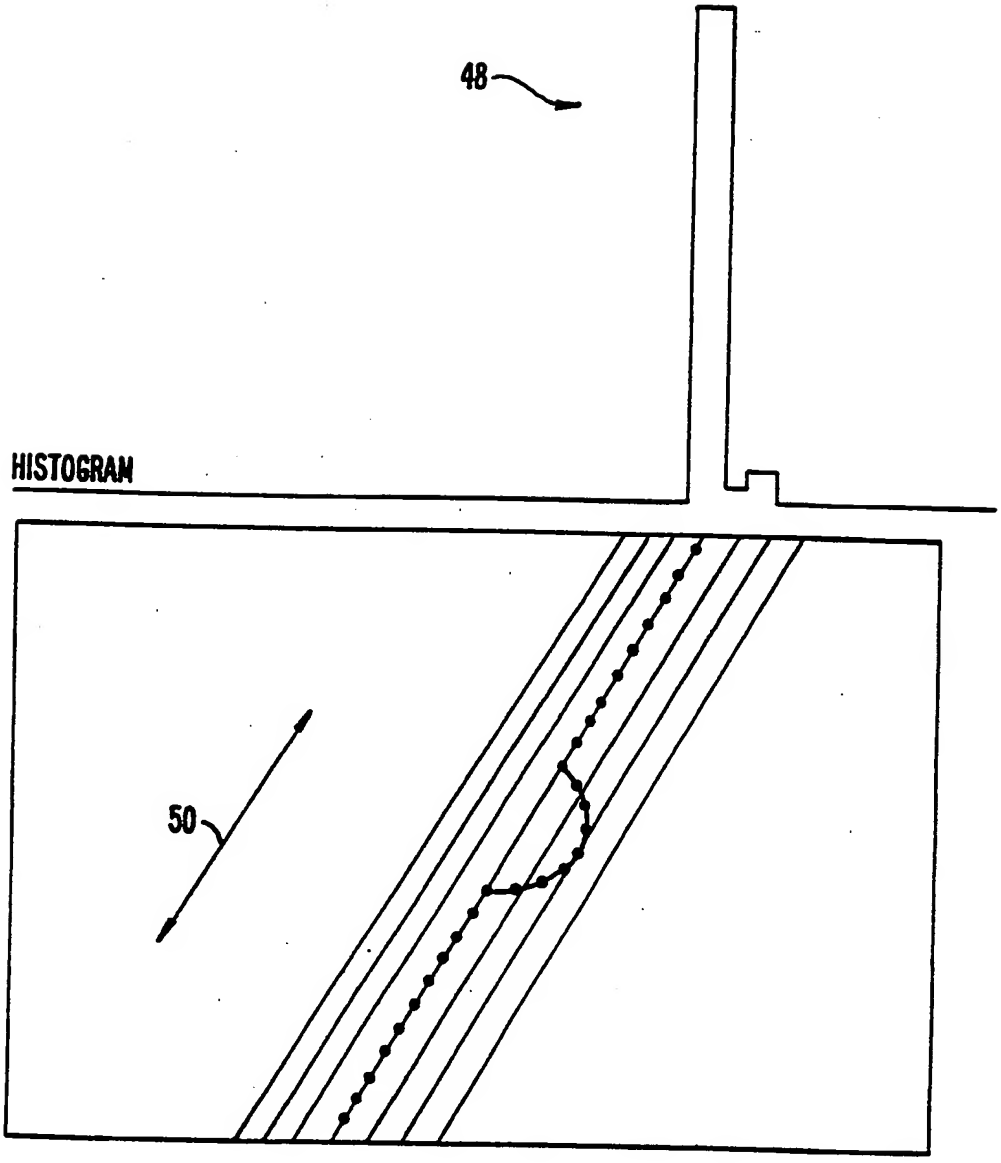


FIG. 4C.

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US91/09477

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) *

According to International Patent Classification (IPC) or to both National Classification and IPC

US CL : 382/22

IPC(5): G06K 9/00, 9/46, 9/48, 9/34, 9/36

II. FIELDS SEARCHED

Minimum Documentation Searched 7

Classification System	Classification Symbols
U.S.	382/1,18,22,24,9,41,56

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched 8

III. DOCUMENTS CONSIDERED TO BE RELEVANT 9

Category *	Citation of Document, 11 with indication, where appropriate, of the relevant passages 12	Relevant to Claim No. 13
A	J. Illingworth et al., "The Adaptive Hough Transform" IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. PAMI-9 No. 5. September 1987, see pages 690-698.	1-18
A	Li, Hungwen et al., "Fast Hough Transform: A Heirarchichal Approach", Computer Vision, Graphics, and Image Processing, 36, 1986. See pages 139-161.	1-18

* Special categories of cited documents: 10

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search

21 February 1992

Date of Mailing of this International Search Report

08 APR 1992

International Searching Authority

ISA/US

Signature of Authorized Officer

MICHAEL R. CAMMARATA

Form PCT/ISA/210 (second sheet) (Rev.11-87)

FURTHER INFORMATION CONTINUED FROM THE SECOND SHEET

V. ☒ OBSERVATIONS WHERE CERTAIN CLAIMS WERE FOUND UNSEARCHABLE [1]

This international search report has not been established in respect of certain claims under Article 17(2) (a) for the following reasons:

1. ☒ Claim numbers 1-18 because they relate to subject matter [2] not required to be searched by this Authority, namely:

Mathematical theory for detecting collinear edge points and measuring distance between two edges comprises claims 1-18 and is not required to be searched by this Authority.

2. ☐ Claim numbers . because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out [3], specifically:

3. ☐ Claim numbers . because they are dependent claims not drafted in accordance with the second and third sentences of PCT Rule 6.4(a).

VI. ☐ OBSERVATIONS WHERE UNITY OF INVENTION IS LACKING [4]

This International Searching Authority found multiple inventions in this international application as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims of the international application.

2. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims of the international application for which fees were paid, specifically claims:

3. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claim numbers:

4. ☐ As all searchable claims could be searched without effort justifying an additional fee, the International Searching Authority did not invite payment of any additional fee.

Remark on Protest

☐ The additional search fees were accompanied by applicant's protest.

☐ No protest accompanied the payment of additional search fees.